**JavaScript编码规范**

1. **JavaScript 文件引用**

JavaScript 程序应该尽量放在 .js 的文件中，需要调用的时候在 HTML 中以 <script src="filename.js"> 的形式包含进来。JavaScript 代码若不是该 HTML 文件所专用的，则应尽量避免在 HTML 文件中直接编写 JavaScript 代码。因为这样会大大增加 HTML 文件的大小，无益于代码的压缩和缓存的使用。

另外，<script src="filename.js"> 标签应尽量放在文件的后面。这样会降低因加载 JavaScript 代码而影响页面中其它组件的加载时间。

1. **代码排版**
2. **行长度**

**每行代码应小于 80 个字符。**如果代码较长，应尽量选择换行，下一行代码应缩进 8 个空格。这样可以使代码排版整齐，减轻阅读代码的疲劳感。换行缩进 8 个空格可以和代码段的缩进 4 个空格区分开，以增强代码的可阅读性。

1. **行结束**

JavaScript 语句应该以分号结束。但大多数浏览器允许不写分号，只要在本应是分号的地方有一个换行符就行。但是如果代码行较长需要换行的时候，有哪些注意事项呢？换行应选择在操作符和标点符号之后，最好是在逗号','之后，而不要在变量名、字符串、数字、或')' ']' '++' '--'等符号之后换行。

这样可以有效的防止拷贝、粘贴而引起的错误，并可有效地增强代码的可阅读性。请见清单 1，代码的输出符合我们的期望。但就写法而言，对 valueB 的赋值语句是在变量 valueA 之后进行的换行，这很容易被误解为 valueB=ValueA，给阅读造成障碍。而对 valueC 的复制语句是在'+'之后进行的换行，就容易理解的多。这也是本文所提倡的换行方式。

**清单 1. 行结束的位置**

<script language="javascript">

var valueA = 1;

var valueB = valueA ///bad

+1;

var valueC = valueB + ///good

valueA;

alert(valueB); //output: valueB=2

alert(valueC);//output: valueC=3

</script>

1. **缩进**

关于缩进的问题，不只是 JavaScript，几乎所有的语言编写的时候，都会提及缩进的问题。缩进几乎是代码编写规范的第一课，是代码可阅读性判断的直接因素。

代码缩进的好处是不言而喻的，但是对于如何缩进，则没有标准而言。最受欢迎的是方便使用 TAB 键缩进，也有些喜欢用 2 个、4 个、8 个空格进行缩进。这样缩进风格不一，也同样给代码的阅读带来障碍。

本文提倡用 **4 个空格**来进行缩进，并在同一产品中采用同一种缩进标准。不支持用 TAB 键进行缩进。这是因为直到现在还没有统一的标准来定义 TAB 键所代替的空白大小，有些编辑器解析为 4 个空格大小，有些则解析为 8 个。因而用不同的编辑器查看代码，可能造成格式混乱。当然 TAB 简单易用，为解决这个问题，建议在设置开发环境时，将编辑器里的 TAB 快捷键重新设置为 4 个空格。据了解 Eclipse, Vi, Nodepad++，Editplus, UltraEdit 等流行的编辑器，均提供了此功能。

1. **注释**

代码中的注释很重要，自然也是毋庸置疑的。通常我们会强调代码中注释数量的多少，而轻视了对注释质量的提高。编码是及时添加注释，会给后续代码的维护人员带来很大的便利。但是如果注释不注意更新，或者由于拷贝、粘贴引起的错误的注释，则会误导阅读人员，反而给阅读带来障碍。

除了注释要 **及时更新**外，我们还应对注释的内容要特别关注。注释要尽量简单、清晰明了，避免使用含混晦涩的语言，同时着重 **注释的意义**，对不太直观的部分进行注解。请见清单 2。

**清单 2. 有意义的注释**

<script language="javascript">

//following section is used to initialize golbal variables (good)

var valueA = 0; //initialize valueA to be sero (bad)

var valueB = 1;

...

//call f1 function after waiting for 50 seconds. (good)

setTimeout(f1,50000); //set timeout to be 20s (copy error)

...

</script>

这样的注释方式在 JavaScript 代码中经常见到。"initialize valueA to be sero" 这样的注释有什么用呢？难道阅读程序的工程师从"var valueA = 0;"复制语句中看不出来么？"set timeout to be 20s"这条注释，不只是因拷贝、粘贴引起的时间大小的错误，同时也误导了程序员对这条语句的理解。setTimeout() 函数的作用并非是设置函数执行的超时时间，而是等待一定时间后执行所调用的函数，害人匪浅呀。这样的注释内容宁可删掉。

此外，JavaScript 的注释有两种"//" 和"/\* .... \*/"，建议"//"用作代码行注释，"/\* .... \*/"形式用作对整个代码段的注销，或较正式的声明中，如函数参数、功能、文件功能等的描述中。

1. **标识符命名**

JavaScript 中的标识符的命名规则：

* 以字母、下划线'\_'或美元符号'$'开头
* 允许名称中包含字母，数字，下划线'\_'和美元符号'$'
* 区分大小写

变量、参数、成员变量、函数等名称均以小写字母开头，构造器的名称以大写字母开头。下划线'\_'开头的变量一般习惯于标识私有 / 局部成员。而美元符号'$'开头的变量习惯于标识系统相关，比如系统进程等。应避免用下划线'\_'或美元符号'$'来命名标识符。尽可能地降低代码的阅读负担。

1. **声明**
2. **变量的声明**

尽管 JavaScript 语言并不要求在变量使用前先对变量进行声明。但我们还是应该养成这个好习惯。这样可以比较容易的检测出那些未经声明的变量，避免其变为隐藏的全局变量，造成隐患。

在函数的开始应先用 var 关键字声明函数中要使用的局部变量，注释变量的功能及代表的含义，且应以字母顺序排序。每个变量单独占一行，以便添加注释。这是因为 JavaScript 中只有函数的 {} 表明作用域，用 var 关键字声明的局部变量只在函数内有效，而未经 var 声明的变量则被视为全局变量。我们来看下清单 3。

**清单 3. 局部变量声明**

<script language="javascript">

var valueA = "a";

var valueB = "b";

function f1() {

var valueA = "c";

alert("valueA="+valueA); //output: valueA=c

valueB = "d";

alert("valueB="+valueB); //output: valueB=d

}

f1();

alert("valueA="+valueA); //output: valueA=a

alert("valueB="+valueB); //output: valueB=d

</script>

从上例的输出惊奇地发现，用 var 声明过的变量 valueA 和没有声明的变量 valueB 是有区别的。特别需要注意的是，在函数内部用 var 声明的变量为局部变量，这样可以有效地避免因局部变量和全局变量同名而产生的错误。

1. **函数的声明**

函数也应在调用前进行声明，内部函数应在 var 声明内部变量的语句之后声明，可以清晰地表明内部变量和内部函数的作用域。

此外，函数名紧接左括号'('之间，而右括号')'和后面的'{'之间要有个空格，以清楚地显示函数名以其参数部分，和函数体的开始。若函数为匿名 / 无名函数，则 function 关键字和左括号'('之间要留空格，否则可能误认为该函数的函数名为 function。

**清单 4. 内部函数声明**

<script language="javascript">

var innerA = 1;

function outF() {

var innerA = 2;

function \_inF() {

alert("valueA="+innerA);

}

\_inF();

}

outF(); //output: valueA=2

\_inF(); //error: innerF is not defined

</script>

从清单 4 的输出可以看出，inF() 函数仅在 outF() 函数的内部生效，局部变量 innerA 对内部函数的作用域生效。这样的编码方式使得变量和函数的作用域变得清晰。

1. **语句**

对于简单语句而言，需要提及的仍然是分号必要性，同时，一行最多有一个语句。如果一个赋值语句是用函数和对象来赋值，可能需要跨多行，一定切记要在赋值语句末加上分号。

这是因为 JavaScript 中，所有表达式都可以当语句，遇换行符时会解析为表达式的结束，此时不规范的换行和分号的丢失，可能引入新的错误。

对于复合语句，if, for, while, do, switch, try … catch 等代码体，函数定义的函数体，对象的定义等都需要放在花括号'{}'里面。

* '{' 应在行末，标志代码块的开始。
* '}' 应在一行开头，标志代码块的结束，同时需要和'{'所在行的开始对齐，以表明一个完整的复合语句段。这样可以极大地提高代码的可阅读性，控制逻辑能清晰地表现出来。
* 被包含的代码段应该再缩进 4 个空格。
* 即使被包含的代码段只有一句，也应该用花括号'{}'包含。尽管不用花括号代码也不会错，但如若需要增加语句的话，则较容易因花括号遗漏而引起的编译错误或逻辑错误。

**return**语句在使用时也需慎重，如果用表达式的执行作为返回值，请把表达式和 return 放在同一行中，以免换行符被误解析为语句的结束而引起返回错误。return 关键字后若没有返回表达式，则返回 undefined。构造器的默认返回值为 this。

**清单 5. return 表达式**

<script language="javascript">

function F1() {

var valueA = 1;

var valueB = 2;

return valueA + valueB;

}

function F2() {

var valueA = 1;

var valueB = 2;

return

valueA + valueB;

}

alert( F1() ); //output: 3

alert( F2() ); //ouput: undefined

</script>

在清单 5 中显示了因返回表达式没有和 return 关键字放在同一行而引起的返回错误，需重视。

1. **特殊符号**

**空白符**

适当的空白行可以大大提高代码的可阅读性，可以使代码逻辑更清晰易懂。同时，在表达式中适当的留空白，也会给代码的阅读带来方便。

关键字的后面如有括号，则最好在关键字和左括号'('之间留空白，如 for, if, while 等。而函数名和括号之间则不宜留空白，但若是匿名函数，则必须在 function 和左括号'('之间留空白，否则，编辑器会误认为函数名为 function。

在表达式中，二元运算符 ( 除左括号'('，左方括号'['，作用域点'.') 和两个操作数之间最好留空白。一元运算符（若不是词 typeof 等）和其操作数之间不宜留空白。

逗号','的后面需要留空白，以显示明确的参数间隔，变量间隔等。

分号';'之后通常表明表达语句的结束，而应空行。在 for 的条件语句中，分号之后则应该留空白。

**{ } 和 [ ]**

在 JavaScript 中，如需定义空对象和空数组，通常很自然地想到用 new Object() 和 new Array() 的方法。其实花括号'{}'和方括号'[]'可以直接用来定义一个空对象和一个空数组。这种书写方法可以使代码看起来简单易懂。

**== 和 ===**

判断"逻辑等"在代码里太平常的不过事情了，但 JavaScript 与其他熟知的编程语言不同的是，除了可以使用两个等号'=='来作判断以为，还可以使用三个等号'==='来进行逻辑等判断。两者的不同是'=='作逻辑等判断时，会先进行类型转换后再进行比较。'==='则不会。因而，'=='进行的判断结果可能产生偏差。'!='与'!=='的区别亦是如此。本文提倡尽量使用'==='来进行逻辑等的判断，用'!=='进行逻辑不等的判断。

**清单 6. === 的使用**

<script language="javascript">

var valueA = "1";

var valueB = 1;

if ( valueA == valueB) {

alert("Equal");

}

else {

alert("Not equal")

}

//output: "Equal"

if ( valueA === valueB) {

alert("Equal");

}

else {

alert("Not equal")

}

//output: "Not equal"

</script>

清单 6 中，valueA 和 valueB 两个变量的值显然是不相等的，起码 valueA 是个字符串，而 valueB 是一个数字。但用'=='进行判断是，程序却输出相等的字样。这是因为编译器对两个变量进行比较时，因为他们的类型不同，而自动地将 valueB 转换成字符串，而后再和 valueA 进行比较的。用'==='得到的判断结果正和预期的结果相符。

**+**

加号'+'也同样是程序员所熟知的操作符之一。JavaScript 和其他编程语言不同的是，在 JavaScript 中，'+'除了表示数字值相加，字符串相连接以外，还可以作一元运算符用，把字符串转换为数字。因而如果使用不当，则可能与自增符'++'混淆而引起计算错误。这一点，在清单 7 中可以清楚地看出。

**清单 7. 巧用 + 号**

<script language="javascript">

var valueA = 20;

var valueB = "10";

alert( valueA + valueB); //ouput: 2010

alert( valueA + (+valueB)); //output: 30

alert( valueA + +valueB); //output:30

alert( valueA ++valueB); //Compile error

</script>

1. **总结**

本文就 JavaScript 代码的排版、命名、声明、语句、和一些特殊字符的使用等方面，谈了自己对 JavaScript 编程规范的建议。此外，还有许多方面需要深入了解研究，如 with, eval 语句和 this 对象的使用等等。我们在认识其普遍性的同时也需要注意其特殊性，在编写代码时多用心留意，以创造更多更优质的程序代码。